Projekt 3: analiza syntaktyczna i leksykalna

1. Wstęp.

Celem niniejszego projektu jest zaimplementowanie skanera i parsera dla C– przy użyciu generatora skanerów *lex* lub *flex* i generatora parserów *bison.* Dokumentację do *lex*'a/*flex*'a znajdziecie tutaj:
`http://www.kompilatory.agh.edu.pl/pages/tk-laboratorium/flex.html`
a do *bison*'a tutaj:
`http://www.kompilatory.agh.edu.pl/pages/tk-laboratorium/bison.html`
z kolei specyfikację języka C– tutaj:
`http://www.math.us.edu.pl/~pgladki/teaching/2011-2012/tk-cminusminus.html`
Na potrzeby tego zadania zajmować będziecie się tylko regułami leksykalnymi i syntaktycznymi C–. Innymi słowy, wszystko to, co wymaga informacji semantycznej, tj. wykorzystuje deklaracje – powinno zostać zignorowane.

2. Skaner

2.1. **Uwagi ogólne.** Skaner powinien zostać zaimplementowany jako funkcja, która za każdym wywołaniem zwraca albo liczbę naturalną oznaczającą, jakiego rodzaju token został znaleziony w strumieniu wejściowym, albo liczbę zero (koniec pliku) oznaczającą, że nie ma dalszych danych na wejściu. Słowa kluczowe nie mogą zostać użyte jako identyfikatory.

Wartości dla poszczególnych rodzajów tokenów powinny być zdefiniowane jako makra celem uproszczenia interfejsu pomiędzy skanerem i parserem. W tym celu najprościej jest zdefiniować jednoznakowe tokeny takie, jak (i ; aby miały wartość odpowiadającej stałej znakowej, tj. aby wartość tokena "lewego nawiasu" była równa wartości stałej znakowej "(".

2.2. **Komentarze i białe spacje.** Komentarze i białe spacje mają zostać pominięte "po cichu". W szczególności błędem będzie rozpoznanie końca pliku wewnątrz komentarza.

2.3. **Błędy.** Najprostszym sposobem radzenia sobie z błędami leksykalnymi będzie zostawienie ich obsługi parserowi. Można to osiągnąć po prostu przez zwracanie wartości każdego nieznanego znaku do parsera.

3. Parser

3.1. **Uwagi ogólne.** Należy przekształcić gramatykę tak, aby wyeliminować konflikty. Jedyne dopuszczalne konflikty to konflikt pomiędzy przesuwaniem i redukowaniem "złośliwie" zapętlającego się else oraz konflikt pomiędzy przesuwaniem i redukowaniem produkcji błędów.

3.2. **Błędy.** Waszym zadaniem będzie zaimplementowanie obsługi błędów oraz wychodzenia z błędów dla *błędów syntaktycznych.* W szczególności **nie** obejmuje to błędów semantycznych (tj. wszystkiego, co żąda informacji z deklaracji) – tym zajmiemy się w kolejnym projekcie.

Program powinien sobie radzić z błędami w sensowny sposób i zapisywać komunikaty o błędach do pliku *stderr*. Informacja o błędzie powinna być specyficzna i zawierać numer linii tak, aby użytkownik mógł łatwo dotrzeć do błędu. Wychodzenie z błędu powinno pozwolić parserowi zgrabnie wyjść i kontynuować parsowanie danych z wejścia nawet wtedy, gdy zostaną wykryte błędy syntaktyczne.

4. Wywoływanie programu.

Plik wykonywalny powinien się nazywać *compile* i powinien czytać z pliku *stdin* i zapisywać do pliku *stdout*. Komunikaty o błędach powinny być zapisane do pliku *stderr*.